CSE 190 Discussion 5

HW3: CAVE Simulator

Agenda

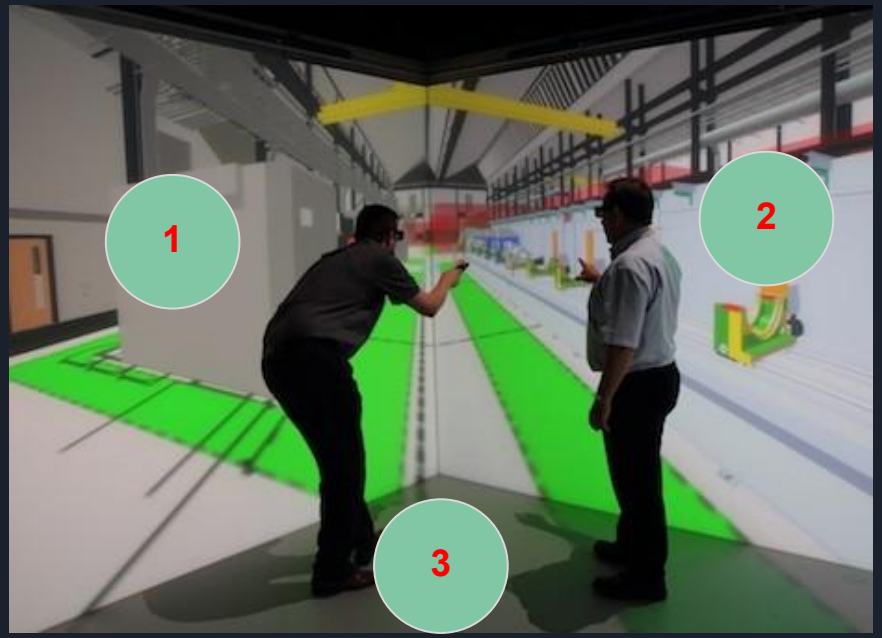
- CAVE Simulator Intro
- Rendering to Texture using OpenGL
- Generalized Perspective Projection
- Helpful references
- Lecture slides

Homework 3 is up

- Link to the assignment: http://ivl.calit2.net/wiki/index.php/Project3S18
- Due Date: May 18th 2pm
  - If you have scheduling conflicts, let us know
- The idea of this assignment:
  - Understand the concept of the CAVE system
  - Learn how to render the scene to textures on quads
  - Figure out the implementation of Perspective Projection
  - And to have fun!

CAVE Simulator

- What you need to do:
  - Render to the scene to the 3 squares respectively
  - Different for each eye
  - Be able to switch the viewport from Head Position to the Controller
  - Be able to freeze the viewport position
  - Manipulate calibration cube



Render Scene To Texture

Render to Texture

- The main idea
  - We need to render different views to different screens.
  - To achieve this, we will
    - Create the texture in which we’re going to render
    - Render each screen as a texture
    - Paste it onto a quad
  - We have three screens for each eye
    - So it is needed to render the scene six times to off-screen buffers

Framebuffers

- Framebuffer is a container for textures
  - It holds textures we can use later
- It allows us to render to places other than the screen we see
- To use the framebuffer:

```
GLuint fbo= 0;

glGenFramebuffers(1, &fbo );

glBindFramebuffer(GL_FRAMEBUFFER, fbo);
```

Textures

- We will need a texture to hold what we are going to draw on our screen

```
GLuint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, TEXTURE_WIDTH, TEXTURE_HEIGHT, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture, 0);
```

Renderbuffers

- We need depth testing but we don’t need to render the depth information
- Renderbuffers are more optimized than textures if you don’t need access

```
GLuint rboId;

glGenRenderbuffers(1, &rboId);
glBindRenderbuffer(GL_RENDERBUFFER, rboId);

glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT,
    TEXTURE_WIDTH, TEXTURE_HEIGHT);
glBindRenderbuffer(GL_RENDERBUFFER, 0);
```

Renderbuffers

- Attach Renderbuffer to framebuffer similarly to our texture

```
glFramebufferRenderbuffer(GL_FRAMEBUFFER,    // 1. fbo target: GL_FRAMEBUFFER
    GL_DEPTH_ATTACHMENT,                     // 2. attachment point
    GL_RENDERBUFFER,                         // 3. rbo target: GL_RENDERBUFFER
    rboId);                                  // 4. rbo ID
```

Rendering to the texture

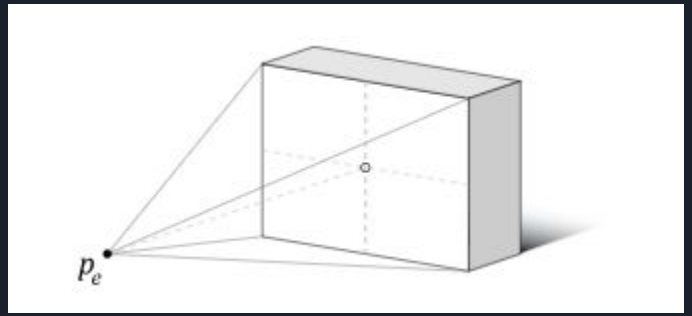
- Just render as normal with the new framebuffer bound
- Then render the resulting texture to a quad that represents the screen with the default framebuffer bound

```
glBindFramebuffer(GL_FRAMEBUFFER,FramebufferName);
```

Generalized Perspective Projection

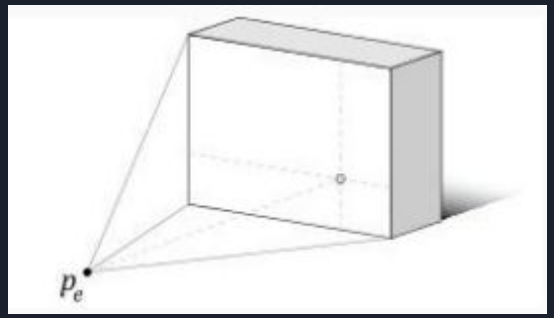
Generalized Perspective Projection

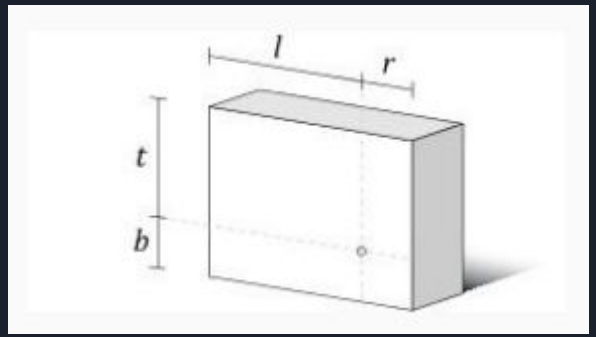
- Typically, the projection matrix we use (generated by gluPerspective) is on-axis



Off-axis perspective

- In a CAVE, we cannot view every screen head on, so each screen needs a different perspective.
- glFrustum can generate the perspective matrix for us given several parameters

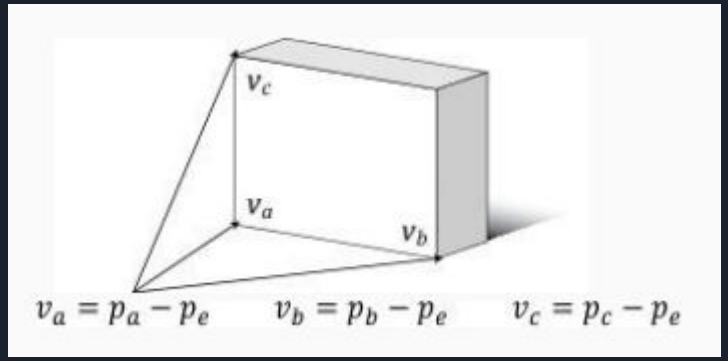




(The near and far plane parameters is not shown)

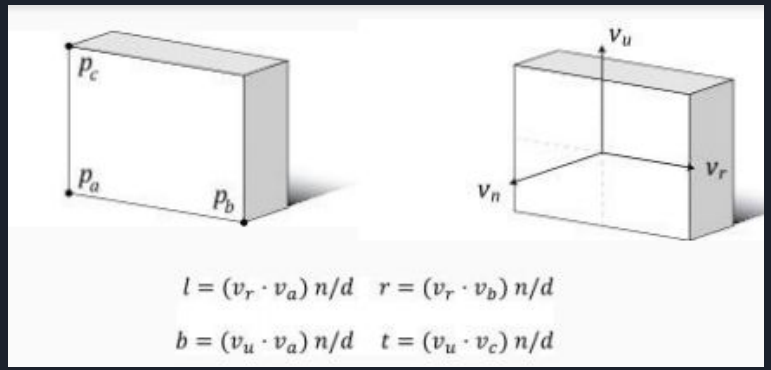
Calculating frustum parameters

1. Find vectors from eye to corners.



2. Find distance from eye to plane

$$d = -(v_n \cdot v_a)$$



3. Find extents, scaling to the near plane

Almost there

- We need to more capabilities:
  - Rotate the screen out of the XY plane
  - Correctly position it relative to the user

Projection Plane Orientation

- We want something lying in screen plane to be transformed to XY plane.
- Use inverse of screen coordinate system (since they are orthogonal we can use transpose)

$$M = \begin{bmatrix} v_{rx} & v_{ux} & v_{nx} & 0 \\ v_{ry} & v_{uy} & v_{ny} & 0 \\ v_{rz} & v_{uz} & v_{nz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

View Point Offset

- Need to account for eye offset

$$T = \begin{bmatrix} 1 & 0 & 0 & -p_{ex} \\ 0 & 1 & 0 & -p_{ey} \\ 0 & 0 & 1 & -p_{ez} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Can be accomplished using the OpenGL function glTranslatef

Putting Everything Together

- Finally, all put together

$$P' = PM^T T$$

- A sample implementation of the perspective matrix
  - http://csc.lsu.edu/~kooima/articles/genperspective/

Helpful References

- Framebuffers
  - https://learnopengl.com/Advanced-OpenGL/Framebuffers
  - http://www.songho.ca/opengl/gl_fbo.html
- Render to Texture
  - http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/
- Generalized Perspective Projection
  - http://csc.lsu.edu/~kooima/articles/genperspective/

QUESTIONS?